Kernel concurrency bugs

CSE 451, 2015

Pedro Fonseca
Concurrency bugs

- Depend on the interleaving of instructions
  - Non-deterministic

- Hard to avoid concurrency in the multi-core era
  - Finer grained locking
  - New algorithms

- Can have serious consequences
  - Therac-25 accident
Concurrent concurrency bugs

```c
int x = 0;

void threadA(){
    A = x + 1;
    x = A;
}

void threadB(){
    B = x + 1;
    x = B;
}
```
Concurrent bug example

```c
int x = 0;

void threadA()
{
    A = x + 1;
    x = A;
}

void threadB()
{
    B = x + 1;
    x = B;
}
```
Concurrency bug example

```c
int x = 0;
void threadA(){
    A = x + 1;
    x = A;
}
void threadB(){
    B = x + 1;
    x = B;
}
```

1. A = x + 1
   B = x + 1
   x = A
   x = B
2. A = x + 1
   B = x + 1
   x = A
   x = B
Concurrency bug example

int x = 0;

void threadA()
{
    A = x + 1;
    x = A;
}

void threadB()
{
    B = x + 1;
    x = B;
}
Concurrent bug example

**How many interleavings?**

```c
int x = 0;

void threadA()
{
    A = x + 1;
    x = A;
}

void threadB()
{
    B = x + 1;
    x = B;
}
```

1. 
   ```c
   A = x + 1
   x = A
   ```

2. 
   ```c
   A = x + 1
   x = A
   ```

3. 
   ```c
   A = x + 1
   x = B
   ```
Concurrency bug example

How many interleavings?

\[ \binom{a + b}{a! \times b!} \]

\(a, b \rightarrow \text{number of instructions}\)
Concurrence bug example

How many interleavings?

\[
\frac{(a + b)!}{a! \times b!}
\]

\(a, b \rightarrow \text{number of instructions}\)

Too many!!
Only a subset of executions exposes

- Often the subset of executions is **really tiny**
  - Concurrency bugs may go unnoticed for **years**!

- Small environment differences may expose them more easily
  - Different kernels, different libraries, different workloads, different hardware
  - Bugs may never show up during testing and always show up on users computers!

- Non-determinism can be really painful!
Concurrency bugs can have many effects:

- Crash
- Hang
- Wrong results

\[ 1 + 1 = 3 \]
Concurrent bug example

```c
int x = 0;

void threadA() {
    Lock_x.acquire();
    A = x + 1;
    x = A;
    Lock_x.release();
}

void threadB() {
    Lock_x.acquire();
    B = x + 1;
    x = B;
    Lock_x.release();
}
```
Concurrent bug example

```c
int x = 0;

void threadA() {
    Lock_x.acquire();
    A = x + 1;
    x = A;
    Lock_x.release();
}

void threadB() {
    Lock_x.acquire();
    B = x + 1;
    x = B;
    Lock_x.release();
}
```
Concurrent edge example

```c
int x = 0;

void threadA(){
    Lock_x.acquire();
    A = x + 1;
    x = A;
    Lock_x.release();
}

void threadB(){
    Lock_x.acquire();
    B = x + 1;
    x = B;
    Lock_x.release();
}
```

1. $A = x + 1$
   $B = x + 1$
   $x = A$

2. $A = x + 1$
   $B = x + 1$
   $x = A$

3. $A = x + 1$
   $B = x + 1$
   $x = B$
What about this solution?

```c
int x = 0;

void threadA()
{
    x = x + 1;
}

void threadB()
{
    x = x + 1;
}
```
What about this solution?

```c
int x = 0;

void threadA(){
    x = x + 1;
}

void threadB(){
    x = x + 1;
}
```

Still buggy!
What about this solution?

One C statement → Many instructions

00000000004004ed threadA
4004ed:  55
4004ee:  48 89 e5
4004f1:  8b 05 45 0b 20 00
4004f7:  83 c0 01
4004fa:  89 05 3c 0b 20 00
400500:  5d
400501:  c3

push   %rbp
mov    %rsp,%rbp
mov    0x200b45(rip),%eax  # 60103c <x>
add    $0x1,%eax
mov    %eax,0x200b3c(rip)  # 60103c <x>
pop    %rbp
retq

0000000000400502 threadB
400502:  55
400503:  48 89 e5
400506:  8b 05 30 0b 20 00
40050c:  83 c0 01
40050f:  89 05 27 0b 20 00
400515:  5d
400516:  c3

push   %rbp
mov    %rsp,%rbp
mov    0x200b30(rip),%eax  # 60103c <x>
add    $0x1,%eax
mov    %eax,0x200b27(rip)  # 60103c <x>
pop    %rbp
retq
What about this code?

int x = 0;

void threadA()
{
    x = CONSTANT_A;
}

void threadB()
{
    x = CONSTANT_B;
}
What about this code?

int x = 0;

void threadA()
{
    x = CONSTANT_A;
}

void threadB()
{
    x = CONSTANT_B;
}

Still not a good idea!
What about this code?

```
int x = 0;

void threadA()
{
    x = CONSTANT_A;
}

void threadB()
{
    x = CONSTANT_B;
}
```

Still not a good idea!

- Compiler might still emit multiple instructions
What about this code?

```
int x = 0;

void threadA(){
    x = CONSTANT_A;
}

void threadB(){
    x = CONSTANT_B;
}
```

Still not a good idea!

a) Compiler might still emit multiple instructions

and...

b) Some instructions are not atomic
What about this code?

```
int x = 0;

void threadA()
{
    x = CONSTANT_A;
}

void threadB()
{
    x = CONSTANT_B;
}
```

Still not a good idea!

a) Compiler might still emit multiple instructions and...

b) Some instructions are not atomic and...

c) C standard says don't do it
What about this code?

```c
int x = 0;

void threadA()
{
    x = CONSTANT_A;
}

void threadB()
{
    x = CONSTANT_B;
}
```

Still not a good idea!*

a) Compiler might still emit multiple instructions and...

b) Some instructions are not atomic and...

c) C standard says don't do it

* Kernel developers sometimes do it though
Kernel concurrency bugs

- Bugs that depend on the instruction interleavings
  - **Triggered only by a subset** of the interleavings
- Plenty of kernel concurrency bugs in kernels!

---

Linux 3.0.41 change log

[The bug] was quite hard to decode as the reproduction time is between 2 days and 3 weeks and intrusive tracing makes it less likely [...]

Linux kernel mailing list (5/1/2013)

Three of the five 3.4.9 machines [...] **locked up.**

I've tried reproducing the issue, but so far I've been unsuccessful [...]

---

Linux 3.4.41 change log

Three of the five 3.4.9 machines [...] **locked up.**

I've tried reproducing the issue, but so far I've been unsuccessful [...]

---
Approaches to explore interleavings

• Stress testing approach
  – Hope to find the interleaving

• Systematic approach
  – Take full control of the interleavings
  – Existing tools focus on user-mode applications

Focus on operating system kernels
SKI
Finding kernel concurrency bugs

- Testing applications versus kernels
- Our approach
- Implementation
- Evaluation
User-mode tools

Kernel Concurrency Bugs

Previous user-mode systematic tools
LD_PRELOAD, ptrace

App

Kernel

Kernel-level abstractions
Threads and sync. objects
User-mode tools

Previous user-mode systematic tools
LD_PRELOAD, ptrace
Kernel-mode challenges

- Kernel doesn't have a good instrumentation interface

- An alternative would be to modify the kernel
  - But kernel modifications:
    - Change the tested software
    - Are non-trivial
    - Hinder portability

Avoid kernel modifications
User-mode versus kernel-mode

Kernel Concurrency Bugs

App

Kernel-level abstractions
Threads and sync. objects

Kernel

Scheduler

Kernel testing tool

Hardware

Existing user-mode systematic tools
LD_PRELOAD, ptrace

Our tool
(modified VMM)
SKI
Finding kernel concurrency bugs

Systematic
Full control of the kernel interleavings

Practical
No modifications to the kernel
Fast
SKI
Finding kernel concurrency bugs

- Challenges testing the kernel code
- SKI's approach
- Implementation
- Evaluation
SKI's approach

Challenges
1. How to control the schedules?
2. Which contexts are schedulable?
3. Which schedules to choose?
1. How to control the kernel schedules?

- Pin each tested thread to a different CPU (thread affinity)
- Pause and resume CPUs to control schedules

Leverage thread affinity and control CPUs
2. Which contexts are schedulable?

- Execution of some instructions are good hints
- Memory access patterns can also provide hints

Rely on VMM introspection
3. Which schedules to choose?

- PCT: User-mode scheduling algorithm [ASPLOS'10]
  - Run the highest priority live threads
  - Create schedule diversity
- Generalize with interrupt support
  - Detect arrival / end
  - Control dispatch
- Reduce interleaving space

Generalize user-mode systematic testing algorithms
SKI
Finding kernel concurrency bugs

• Challenges testing kernel code
• SKI's approach
• Implementation
• Evaluation
Implementation

- Implemented SKI by modifying QEMU (VMM)
  - No kernel changes required
- Built a user-mode library (VM)
  - Flags start/end of tests and sends results to VMM
  - Used library to implement several test-cases
    - e.g., file system tests
- Implemented several optimizations
Detecting and diagnosing bugs with SKI

- SKI supports different types of bug detectors
  - Crash and assertion violations
  - Data races
  - Semantic bugs (e.g. disk corruption)
- SKI produces detailed execution traces
SKI
Finding kernel concurrency bugs

• Challenges testing kernel code
• SKI's approach
• Implementation
• **Evaluation**
  
  1. Regression testing
  2. Finding previously unknown bugs
1. Regression testing: setup

- Searched for previously reported bugs
  - In kernel bugzilla, mailing lists, git logs
  - Well documented reports and diverse set of bugs
- Created SKI test suites for these bugs
  - By adapting the stress tests in the bug reports
# 1. Regression testing: results

<table>
<thead>
<tr>
<th>Bug</th>
<th>Kernel</th>
<th>Component</th>
<th>Detector</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Linux 2.6.28</td>
<td>Anonymous pipes</td>
<td>Crash</td>
</tr>
<tr>
<td>B</td>
<td>Linux 3.2</td>
<td>Inotify + FAT32</td>
<td>Crash</td>
</tr>
<tr>
<td>C</td>
<td>Linux 3.6.1</td>
<td>Proc + Ext4</td>
<td>Semantic</td>
</tr>
<tr>
<td>D</td>
<td>FreeBSD 8.0</td>
<td>Sockets</td>
<td>Semantic</td>
</tr>
</tbody>
</table>
1. Regression testing: results

<table>
<thead>
<tr>
<th>Bug</th>
<th>Kernel</th>
<th>Component</th>
<th>Detector</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Linux 2.6.28</td>
<td>Anonymous pipes</td>
<td>Crash</td>
</tr>
<tr>
<td>B</td>
<td>Linux 3.2</td>
<td>Inotify + FAT32</td>
<td>Crash</td>
</tr>
<tr>
<td>C</td>
<td>Linux 3.6.1</td>
<td>Proc + Ext4</td>
<td>Semantic</td>
</tr>
<tr>
<td>D</td>
<td>FreeBSD 8.0</td>
<td>Sockets</td>
<td>Semantic</td>
</tr>
</tbody>
</table>
1. Regression testing: results

<table>
<thead>
<tr>
<th>Bug</th>
<th>Kernel</th>
<th>Component</th>
<th>Detector</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Linux 2.6.28</td>
<td>Anonymous pipes</td>
<td>Crash</td>
</tr>
<tr>
<td>B</td>
<td>Linux 3.2</td>
<td>Inotify + FAT32</td>
<td>Crash</td>
</tr>
<tr>
<td>C</td>
<td>Linux 3.6.1</td>
<td>Proc + Ext4</td>
<td>Semantic</td>
</tr>
<tr>
<td>D</td>
<td>FreeBSD 8.0</td>
<td>Sockets</td>
<td>Semantic</td>
</tr>
</tbody>
</table>
## 1. Regression testing: results

<table>
<thead>
<tr>
<th>Bug</th>
<th>Kernel</th>
<th>Component</th>
<th>Detector</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Linux 2.6.28</td>
<td>Anonymous pipes</td>
<td>Crash</td>
</tr>
<tr>
<td>B</td>
<td>Linux 3.2</td>
<td>Inotify + FAT32</td>
<td>Crash</td>
</tr>
<tr>
<td>C</td>
<td>Linux 3.6.1</td>
<td>Proc + Ext4</td>
<td>Semantic</td>
</tr>
<tr>
<td>D</td>
<td>FreeBSD 8.0</td>
<td>Sockets</td>
<td>Semantic</td>
</tr>
</tbody>
</table>

SKI is portable
## 1. Regression testing: results

<table>
<thead>
<tr>
<th>Bug</th>
<th>Kernel</th>
<th>Component</th>
<th>Detector</th>
<th>Schedules</th>
<th>Throughput (sched/h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Linux 2.6.28</td>
<td>Anonymous pipes</td>
<td>Crash</td>
<td>28</td>
<td>302,000</td>
</tr>
<tr>
<td>B</td>
<td>Linux 3.2</td>
<td>Inotify + FAT32</td>
<td>Crash</td>
<td>53</td>
<td>169,300</td>
</tr>
<tr>
<td>C</td>
<td>Linux 3.6.1</td>
<td>Proc + Ext4</td>
<td>Semantic</td>
<td>51</td>
<td>218,700</td>
</tr>
<tr>
<td>D</td>
<td>FreeBSD 8.0</td>
<td>Sockets</td>
<td>Semantic</td>
<td>3519</td>
<td>501,400</td>
</tr>
</tbody>
</table>
## 1. Regression testing: results

SKI can expose bugs in seconds

<table>
<thead>
<tr>
<th>Bug</th>
<th>Kernel</th>
<th>Component</th>
<th>Detector</th>
<th>Schedules</th>
<th>Throughput (sched/h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Linux 2.6.28</td>
<td>Anonymous pipes</td>
<td>Crash</td>
<td>28</td>
<td>302,000</td>
</tr>
<tr>
<td>B</td>
<td>Linux 3.2</td>
<td>Inotify + FAT32</td>
<td>Crash</td>
<td>53</td>
<td>169,300</td>
</tr>
<tr>
<td>C</td>
<td>Linux 3.6.1</td>
<td>Proc + Ext4</td>
<td>Semantic</td>
<td>51</td>
<td>218,700</td>
</tr>
<tr>
<td>D</td>
<td>FreeBSD 8.0</td>
<td>Sockets</td>
<td>Semantic</td>
<td>3519</td>
<td>501,400</td>
</tr>
</tbody>
</table>
1. Regression testing: results

<table>
<thead>
<tr>
<th>Bug</th>
<th>Kernel</th>
<th>Component</th>
<th>Detector</th>
<th>Schedules</th>
<th>Throughput (sched/h)</th>
<th>Schedules</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Linux 2.6.28</td>
<td>Anonymous pipes</td>
<td>Crash</td>
<td>28</td>
<td>302,000</td>
<td>NA (&gt;24h)</td>
</tr>
<tr>
<td>B</td>
<td>Linux 3.2</td>
<td>Inotify + FAT32</td>
<td>Crash</td>
<td>53</td>
<td>169,300</td>
<td>200,000 (4h)</td>
</tr>
<tr>
<td>C</td>
<td>Linux 3.6.1</td>
<td>Proc + Ext4</td>
<td>Semantic</td>
<td>51</td>
<td>218,700</td>
<td>800 (1 min)</td>
</tr>
<tr>
<td>D</td>
<td>FreeBSD 8.0</td>
<td>Sockets</td>
<td>Semantic</td>
<td>3519</td>
<td>501,400</td>
<td>NA (&gt;24h)</td>
</tr>
</tbody>
</table>
1. Regression testing: results

Some stress tests were ineffective

<table>
<thead>
<tr>
<th>Bug</th>
<th>Kernel</th>
<th>Component</th>
<th>Detector</th>
<th>Schedules</th>
<th>Throughput (sched/h)</th>
<th>Schedules</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Linux 2.6.28</td>
<td>Anonymous pipes</td>
<td>Crash</td>
<td>28</td>
<td>302,000</td>
<td>NA (&gt;24h)</td>
</tr>
<tr>
<td>B</td>
<td>Linux 3.2</td>
<td>Inotify + FAT32</td>
<td>Crash</td>
<td>53</td>
<td>169,300</td>
<td>200,000 (4h)</td>
</tr>
<tr>
<td>C</td>
<td>Linux 3.6.1</td>
<td>Proc + Ext4</td>
<td>Semantic</td>
<td>51</td>
<td>218,700</td>
<td>800 (1 min)</td>
</tr>
<tr>
<td>D</td>
<td>FreeBSD 8.0</td>
<td>Sockets</td>
<td>Semantic</td>
<td>3519</td>
<td>501,400</td>
<td>NA (&gt;24h)</td>
</tr>
</tbody>
</table>
2. Finding previously unknown bugs

- Created a SKI test suit for file systems
  - Adapted the existing *fsstress* test suit
  - Tested several file systems
- Bug detectors
  - Crashes, warnings, data races, semantic errors (*fsck*)
- Tested recent versions of Linux
## 2. Finding previously unknown bugs

<table>
<thead>
<tr>
<th>Bug</th>
<th>Linux</th>
<th>FS</th>
<th>Detector / Failure</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.11.1</td>
<td>Btrfs</td>
<td>Crash (Null-pointer)</td>
<td>Fixed</td>
</tr>
<tr>
<td>2</td>
<td>3.11.1</td>
<td>Btrfs</td>
<td>Crash (Null-pointer) + Warning</td>
<td>Fixed</td>
</tr>
<tr>
<td>3</td>
<td>3.11.1</td>
<td>Btrfs</td>
<td>Warning</td>
<td>Fixed</td>
</tr>
<tr>
<td>4</td>
<td>3.11.1</td>
<td>Btrfs</td>
<td>Fsck (References not found)</td>
<td>Reported</td>
</tr>
<tr>
<td>5</td>
<td>3.11.1+p</td>
<td>Btrfs</td>
<td>Crash (Null-pointer)</td>
<td>Fixed</td>
</tr>
<tr>
<td>6</td>
<td>3.12.2</td>
<td>Btrfs</td>
<td>Warning</td>
<td>Fixed</td>
</tr>
<tr>
<td>7</td>
<td>3.13.5</td>
<td>Logfs</td>
<td>Crash (Null-pointer)</td>
<td>Reported</td>
</tr>
<tr>
<td>8</td>
<td>3.13.5</td>
<td>Logfs</td>
<td>Crash (Invalid paging)</td>
<td>Reported</td>
</tr>
<tr>
<td>9</td>
<td>3.13.5</td>
<td>Jfs</td>
<td>Crash (Assertion violation)</td>
<td>Reported</td>
</tr>
<tr>
<td>10</td>
<td>3.13.5</td>
<td>Ext4</td>
<td>Data race</td>
<td>Fixed</td>
</tr>
<tr>
<td>11</td>
<td>3.13.5</td>
<td>VFS</td>
<td>Data race</td>
<td>Reported</td>
</tr>
</tbody>
</table>
### 2. Finding previously unknown Linux bugs

<table>
<thead>
<tr>
<th>Bug</th>
<th>Linux</th>
<th>FS</th>
<th>Detector / Failure</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.11.1</td>
<td>Btrfs</td>
<td>Crash (Null-pointer)</td>
<td>Fixed</td>
</tr>
<tr>
<td>2</td>
<td>3.11.1</td>
<td>Btrfs</td>
<td>Crash (Null-pointer) + Warning</td>
<td>Fixed</td>
</tr>
<tr>
<td>3</td>
<td>3.11.1</td>
<td>Btrfs</td>
<td>Warning</td>
<td>Fixed</td>
</tr>
<tr>
<td>4</td>
<td>3.11.1</td>
<td>Btrfs</td>
<td>Fsck (References not found)</td>
<td>Reported</td>
</tr>
<tr>
<td>5</td>
<td>3.11.1+p</td>
<td>Btrfs</td>
<td>Crash (Null-pointer)</td>
<td>Fixed</td>
</tr>
<tr>
<td>6</td>
<td>3.12.2</td>
<td>Btrfs</td>
<td>Warning</td>
<td>Fixed</td>
</tr>
<tr>
<td>7</td>
<td>3.13.5</td>
<td>Logfs</td>
<td>Crash (Null-pointer)</td>
<td>Reported</td>
</tr>
<tr>
<td>8</td>
<td>3.13.5</td>
<td>Logfs</td>
<td>Crash (Invalid paging)</td>
<td>Reported</td>
</tr>
<tr>
<td>9</td>
<td>3.13.5</td>
<td>Jfs</td>
<td>Crash (Assertion violation)</td>
<td>Reported</td>
</tr>
<tr>
<td>10</td>
<td>3.13.5</td>
<td>Ext4</td>
<td>Data race</td>
<td>Fixed</td>
</tr>
<tr>
<td>11</td>
<td>3.13.5</td>
<td>VFS</td>
<td>Data race</td>
<td>Reported</td>
</tr>
</tbody>
</table>

**Official Linux releases**
2. Finding previously unknown bugs

<table>
<thead>
<tr>
<th>Bug</th>
<th>Linux</th>
<th>FS</th>
<th>Detector / Failure</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.11.1</td>
<td>Btrfs</td>
<td>Crash (Null-pointer)</td>
<td>Fixed</td>
</tr>
<tr>
<td>2</td>
<td>3.11.1</td>
<td>Btrfs</td>
<td>Crash (Null-pointer) + Warning</td>
<td>Fixed</td>
</tr>
<tr>
<td>3</td>
<td>3.11.1</td>
<td>Btrfs</td>
<td>Warning</td>
<td>Fixed</td>
</tr>
<tr>
<td>4</td>
<td>3.11.1</td>
<td>Btrfs</td>
<td>Fsck (References not found)</td>
<td>Reported</td>
</tr>
<tr>
<td>5</td>
<td>3.11.1+p</td>
<td>Btrfs</td>
<td>Crash (Null-pointer)</td>
<td>Fixed</td>
</tr>
<tr>
<td>6</td>
<td>3.12.2</td>
<td>Btrfs</td>
<td>Warning</td>
<td>Fixed</td>
</tr>
<tr>
<td>7</td>
<td>3.13.5</td>
<td>Logfs</td>
<td>Crash (Null-pointer)</td>
<td>Reported</td>
</tr>
<tr>
<td>8</td>
<td>3.13.5</td>
<td>Logfs</td>
<td>Crash (Invalid paging)</td>
<td>Reported</td>
</tr>
<tr>
<td>9</td>
<td>3.13.5</td>
<td>Jfs</td>
<td>Crash (Assertion violation)</td>
<td>Reported</td>
</tr>
<tr>
<td>10</td>
<td>3.13.5</td>
<td>Ext4</td>
<td>Data race</td>
<td>Fixed</td>
</tr>
<tr>
<td>11</td>
<td>3.13.5</td>
<td>VFS</td>
<td>Data race</td>
<td>Reported</td>
</tr>
</tbody>
</table>

Requested by developers

- Bug 1
- Bug 2
- Bug 4
- Bug 5

Finding previously unknown bugs requested by developers.
## 2. Finding previously unknown bugs

<table>
<thead>
<tr>
<th>Bug</th>
<th>Linux</th>
<th>FS</th>
<th>Detector / Failure</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.11.1</td>
<td>Btrfs</td>
<td>Crash (Null-pointer)</td>
<td>Fixed</td>
</tr>
<tr>
<td>2</td>
<td>3.11.1</td>
<td>Btrfs</td>
<td>Crash (Null-pointer) + Warning</td>
<td>Fixed</td>
</tr>
<tr>
<td>3</td>
<td>3.11.1</td>
<td>Btrfs</td>
<td>Warning</td>
<td>Fixed</td>
</tr>
<tr>
<td>4</td>
<td>3.11.1</td>
<td>Btrfs</td>
<td>Fsck (References not found)</td>
<td>Reported</td>
</tr>
<tr>
<td>5</td>
<td>3.11.1+p</td>
<td>Btrfs</td>
<td>Crash (Null-pointer)</td>
<td>Fixed</td>
</tr>
<tr>
<td>6</td>
<td>3.12.2</td>
<td>Btrfs</td>
<td>Warning</td>
<td>Fixed</td>
</tr>
<tr>
<td>7</td>
<td>3.13.5</td>
<td>Logfs</td>
<td>Crash (Null-pointer)</td>
<td>Reported</td>
</tr>
<tr>
<td>8</td>
<td>3.13.5</td>
<td>Logfs</td>
<td>Crash (Invalid paging)</td>
<td>Reported</td>
</tr>
<tr>
<td>9</td>
<td>3.13.5</td>
<td>Jfs</td>
<td>Crash (Assertion violation)</td>
<td>Reported</td>
</tr>
<tr>
<td>10</td>
<td>3.13.5</td>
<td>Ext4</td>
<td>Data race</td>
<td>Fixed</td>
</tr>
<tr>
<td>11</td>
<td>3.13.5</td>
<td>VFS</td>
<td>Data race</td>
<td>Reported</td>
</tr>
</tbody>
</table>

### Important file systems
- Btrfs
- Ext4
- Logfs
- Jfs
- VFS
## 2. Finding previously unknown bugs

<table>
<thead>
<tr>
<th>Bug</th>
<th>Linux</th>
<th>FS</th>
<th>Detector / Failure</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.11.1</td>
<td>Btrfs</td>
<td>Crash (Null-pointer)</td>
<td>Fixed</td>
</tr>
<tr>
<td>2</td>
<td>3.11.1</td>
<td>Btrfs</td>
<td>Crash (Null-pointer) + Warning</td>
<td>Fixed</td>
</tr>
<tr>
<td>3</td>
<td>3.11.1</td>
<td>Btrfs</td>
<td>Warning</td>
<td>Fixed</td>
</tr>
<tr>
<td>4</td>
<td>3.11.1</td>
<td>Btrfs</td>
<td>Fsck (References not found)</td>
<td>Reported</td>
</tr>
<tr>
<td>5</td>
<td>3.11.1+p</td>
<td>Btrfs</td>
<td>Crash (Null-pointer)</td>
<td>Fixed</td>
</tr>
<tr>
<td>6</td>
<td>3.12.2</td>
<td>Btrfs</td>
<td>Warning</td>
<td>Fixed</td>
</tr>
<tr>
<td>7</td>
<td>3.13.5</td>
<td>Logfs</td>
<td>Crash (Null-pointer)</td>
<td>Reported</td>
</tr>
<tr>
<td>8</td>
<td>3.13.5</td>
<td>Logfs</td>
<td>Crash (Invalid paging)</td>
<td>Reported</td>
</tr>
<tr>
<td>9</td>
<td>3.13.5</td>
<td>Jfs</td>
<td>Crash (Assertion violation)</td>
<td>Reported</td>
</tr>
<tr>
<td>10</td>
<td>3.13.5</td>
<td>Ext4</td>
<td>Data race</td>
<td>Fixed</td>
</tr>
<tr>
<td>11</td>
<td>3.13.5</td>
<td>VFS</td>
<td>Data race</td>
<td>Reported</td>
</tr>
</tbody>
</table>

**Data loss**
Current limitations and future work

• Bugs in kernel scheduler code
  - SKI pins tested threads
    → Represent a small set of bugs

• Bugs in device drivers
  - SKI supports a large set of devices but not all
    → Implement SKI with binary instrumentation techniques

• Bugs that depend on weak memory models
  - SKI currently implements a strong memory model
    → Generalize SKI to also expose these bugs
SKI: Finding kernel concurrency bugs

SKI is Systematic
Full control of the kernel interleavings

SKI is Practical
No modifications to the kernel
Fast

SKI is Effective
Finds and reproduces real-world kernel concurrency bugs

SKI: Exposing Kernel Concurrency Bugs through Systematic Schedule Exploration
Pedro Fonseca, Rodrigo Rodrigues and Björn B. Brandenburg
OSDI'14
Take-away: concurrency is hard!